NQP Final Programming Exercise
– Exact Diagonalisation –

[Summer Semester 2023]

Simon Krumm and Yudong Sun

21st July 2023

1 Introduction

The problem at hand is to investigate the properties of the ground state of a hardcore-bosonic wheel with different number of lattice sites L on the ring (Figure 1.1) using exact diagonalisation (ED) methods.

In this problem, two ways to find the ground state (energies) were proposed:

(i) The ED of a $(L+1) \times (L+1)$ matrix in the operator-valued vector basis, and

(ii) The ED of a $2^{L+1} \times 2^{L+1}$ matrix represented in the occupational (Fock) state basis of the tensor-product Hilbert space.

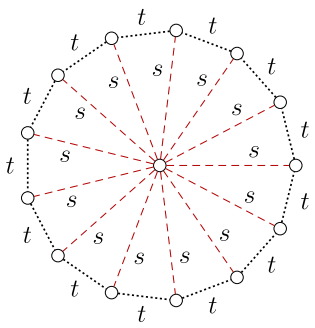

Figure 1.1: Wheel geometry with hopping amplitudes on the ring (t) and from the ring to the center (s). Illustration from the exercise sheet.

An immediate consequence that one notices is that the number of eigenvalues obtained for (i) would be significantly less than that from (ii) simply due to the size of the matrix. The discussion of the comparison between these two methods will be carried out in subsection 3.3.

Before we go into the individual problems, we first describe the general structure of the code. The problem-specific implementation in the code shall then be described in the discussion of the problem itself. The programming language of choice was Python. Certain code segments have also been parallelised with `mpi4py`[1], which uses the Message Passing Interface (MPI) under the hood to allow parallelisation across different computing nodes. Unfortunately, the code does not run properly on the cluster for `nstasks` > 8. This problem will be described further under subsection 2.3.

The personal desktop PC ultimately used for the generation of this data ran on a 8-core, 16-thread Ryzen 7 5800X, which allowed us to run the calculations with `nstasks` $= 16$ at higher processor speed (4.5 GHz). In the end, the data used for the graphs in this report took approximately 6 hours to generate, before the data was processed and plotted.

[1] https://mpi4py.readthedocs.io/en/stable/

2 Overall Implementation and Code Structure

2.1 Scaffolding

The scaffolding of the code is provided by the file `QM.py`, which contains the following classes:

- `State`
- `Operator`
- `HBFockState < State`
- `HBFockOperator < Operator`

where `A < B` means that `A` derives from `B`, or in other words, `B` is the parent class of `A`. `HBFock` comes from *Hardcore Boson Fock*.

Since operators may be represented by either dense or sparse matrices, this structure allows us to do data-structure agnostic diagonlisaton of the hamiltonians without needing to change large swaths of code. One may choose the representation by passing the parameter `sparse = True | False` into the constructor of `(HBFock)Operator`.

Since we are only using states to calculate matrix elements, the time/efficiency benefit of using sparse matrices to represent states are minimal. Therefore, to reduce complexity, all states are represented by dense $(1 \times M)$ or $(M \times 1)$ `numpy` arrays.

All arithmetic operations have also been overloaded such that an expression such as the following becomes possible:

$$\langle\varphi|\,\hat{b}^\dagger\,|\varphi\rangle \qquad \equiv \qquad \texttt{phi.dagger() @ b.dagger() @ phi} \tag{1}$$

where `phi` is a `State` and `b` is an `Operator`.

Since `State` also internally stores whether it represents a ket- or a bra-state, the overloaded operations allow us to quickly check if a specific operation is allowed. For example, $\hat{b}\,|\varphi\rangle \to |\varphi'\rangle$ and $\langle\varphi|\,\hat{b} \to \langle\varphi''|$ are allowed but not $\hat{b}\,\langle\varphi| \to ?$. This specifies the expected behaviour of the code and helps us to prevent certain types of errors in the calculations later on that, for example, involve wrong shapes. As a bonus, we will also know the exact datatype of the resulting object after matrix multiplication in normally ambiguous situations:

$$\langle\varphi|\psi\rangle = (\texttt{State})\ \texttt{@}\ (\texttt{State}) = \text{Number} \tag{2}$$

$$|\varphi\rangle\langle\psi| = (\texttt{State})\ \texttt{@}\ (\texttt{State}) = \texttt{Operator} \tag{3}$$

The classes `HBFockState` and `HBFockOperator` are then a specific implementation of `State` and `Operator` respectively for the problem at hand. They contain checks for shape requirements and stores additional parameters relevant for the calculation. Furthermore, `HBFockOperator` provides a method for expanding single site ($\tilde{L} = 0$, "only center site") operators to the many-body ($\tilde{L} = L$) Hilbert space. There, we make use of the fact that the tensor product of identity matrices is simply a larger identity matrix ($\mathbb{1}_2^{\otimes n} = \mathbb{1}_{2^n}$) to simplify the calculations and reduce overhead.

Where possible, robust type-hinting has also been added to the code. This way, code could be more easily be verfied, and it would be easier for another person to follow the logic in the code.

2.2 Main Code

The main code that does the calculation is contained in `src/main.py` in the form of a `Lattice` class and the function `condensate_frac()`.

The `Lattice` class provides the wrapper for a wheel lattice with a given number of lattice sites L in the ring. It contains the methods for building the Hamiltonian in the following bases:

- Operator-valued vector basis (subsection 3.1, Problem 1(a)):

  `lattice.build_hamiltonian_manual(t, s)` $\rightarrow \underline{\underline{H}}_{\odot}$ = `lattice.hamiltonian["manual"]`

- Fock state basis of the tensor-product Hilbert space (subsection 3.2, Problem 1(b)):

  `lattice.build_hamiltonian_ed(t, s)` $\rightarrow \hat{H}_{\odot}$ = `lattice.hamiltonian["exact"]`

To obtain the spectrum as a function of the relative coupling s/t, we use the function `spectrum(type_)`, where `type_ = "exact" | "manual"`. In this case, it makes sense to set $t = 1$ and then simply loop over s/t values (`s_t[]`), i.e. s is in units of t. In each step, we build the Hamiltonian

```
lattice.build_hamiltonian_*(t = 1, s = s_t[i])
```

and calculate the desired eigenvalues using the helper functions provided by the `(HBFock)Operator`. The eigenvalues are then saved as a CSV file.

Here, when instantiating the `Lattice` class, we have the choice of representing the Hamiltonians either as sparse or dense matrices. While using sparse matrices may be faster, we are unable to extract all the eigenvalues of the matrix due to the ARPACK implementation of the eigensolver [1]. As a result, we only extract the $N-1$ largest eigenvalues (in terms of magnitude, i.e. $-1 > 0.5$) of the $N \times N$ matrix, which should be sufficient for our purposes. This corresponds to the mode `LM` in `scipy.linalg.sparse`'s eigenvalue function(s). To extract all the eigenvalues, packages such as FILTLAN [2] could be used instead.

For each given L, the condensate fraction is then calculated using the `condensate_frac()` function. Initially, this particular function was separated out from the `Lattice` class as it looped over a sequence of L's, $L \in \mathbb{N}$, $L \geq 2$, and calculated the condensate fraction for each L. However, we realized that since we did not know how long each iteration would take, it would make more sense for `condensate_frac()` to calculate each L separately and save the intermediate data. This way, we could interrupt the running code at any time and retain the results that have already been calculated. Due to time constraints, this function was not rewritten to be a part of the `Lattice` class.

For the calculation of the condensate fraction, using sparse matrices resulted in some artefacts in the results. We thus decided to only use dense matrices for the calculation in problem 1(d) (subsection 3.4).

In our final implementation, we used $S = 112$ equidistant samples between $s/t = 10^{-2}$ and $s/t = 10^{1}$ on the logarithmic scale to calculate the spectrum as a function of the relative coupling, and we used $S = 256$ equidistant samples between $s/t = 10^{-3}$ and $s/t = 10^{3}$ on the logarithmic scale to calculate the condensate fraction.

2.3 Parallelisation

As an embarrassingly (or pleasingly) parallel task, the calculation of the eigensystem for different s/t values was parallelized with the help of `mpi4py`. In this case, our `s_t[]` array was split into `NTasks` equal chucks and then distributed. Each node then did its own calculation for its own set of s/t values before all the results are gathered again back to the root node.

The root node then does the rest of the processing before saving the results into a CSV file.

Initially, a contingent of `NTasks` $= 50$ using the `cip` cluster was planned. However, we kept encountering the following error:

```
srun: fatal: SLURM_MEM_PER_CPU, SLURM_MEM_PER_GPU, SLURM_MEM_PER_NODE are
↪  mutually exclusive
```

which we were unable to resolve on our end. We postulate that if we managed to be lucky enough for the job to be assigned to a single computing node, this problem would resolve itself. This was however difficult to test with `NTasks` $= 50$, since every time we requested a job, it ran across multiple nodes. There was just one MPI successful run on the cluster with `NTasks` $= 8$ and explicitly requesting for `cip-cl-compute7`. This job managed to calculate the condensation fraction for $L = 2$ to $L = 10$ in a little over 2 hours[2]. In comparison, it took about approximately 30 minutes on our local desktop machine with `NTASKS` $= 16$.

We therefore came to the conclusion that while we could reduce `NTASKS` for running on the cluster, the cluster does not actually offer any benefits over using our own desktop PC. It was decided that the parallelised code will then be run on a local desktop PC instead of the cluster. For use on the cluster, it is recommended to just run the "single thread" version without parallelisation using MPI.

2.4 Running the code

The code is strictly split into data acquisition and data processing. The data acquisition is covered in `src/main.py`, which is separately controlled by `src/run.py` and `src/run_cluster.py`. The latter is meant to run the data acquisition on the cluster, while the first was designed to run the data acquisition on a private desktop. This decision was made due to the problems with running the MPI parallelization on the cluster as explained in the section 2.3 above.

Running the data acquisition with or without MPI amounts to passing `mpi = True | False` to the `Lattice` class or the `condensate_frac()` function.

The data is then stored in separate .csv files in the data folder and manually moved to new subfolders to differentiate between runs.

There is no need for manual interaction with `main.py`, only with the file `run.py` should be used to adjust which data to gather and for which L's. This is exemplified in the snippet below for a full MPI data acquisition run on a local desktop. (via `run.py`)

```
for L in range(2,12):
    lattice = Lattice(L, "sparse", mpi = True)
    lattice.spectrum("manual")
    lattice.spectrum("exact")
    condensate_frac(L, matrix_type = "dense", mpi = True)
```

Running the file with MPI amounts to calling `mpiexec -n $NUM_NODES python3 run.py`, where `run.py` can be called in the usual way for a non MPI run (note that the arguments in `run.py` need to be adjusted).

The data processing is purely based on the `data_processing.py` file, which includes three functions, `plot_manual()`, `plot_exact()` and `plot_cf()`. These take care of the plots for Problems a), c) and d) in order, i.e. running the data processing file as provided below will evaluate all data.

```
if __name__ == "__main__":
    plot_manual()
    plot_exact(manual_subset = False)
    plot_exact(manual_subset = True)
    plt_cf()
```

[2]The job script for this job is available under `src/run_cluster_mpi.sh`.

The first two functions will create a plot each in the plots folder that contains 10 subplots, for system sizes ranging from $L = 2$ to $L = 11$. The last function will create a plot with two subplots that show the condensation fraction separately for even and odd L.

The argument `manual_subset` processes the same data if set to `True`, but only plots the first 2 subplots. This is favorable because the range of the data for $L = 2$ and $L = 3$ are much smaller than that of larger system sizes. This affords a better resolution to the plot.

Therefore, the only manual interaction with the `data_processing.py` file needed is the modification of the filepaths to change which folders are targeted to draw data from.

`python3 data_processing.py` can be called in the usual way since there is no MPI interaction from this point on.

2.5 Runtimes

Since running MPI parallelised on the cluster did not work, the fastest runtime was achieved on a private desktop. Naturally, the data acquisition is the crucial step in terms of runtime.

For the largest $L = 11$ (12 lattice sites) run, the full runtime for data acquisition amounted to about 6 hours, where 5 of those were dedicated to the $L = 11$ case. Our estimations put the $L = 12$ case to add another $35 - 40$ hours to this, making this an unreasonable goal for a non-parallelised cluster run.

3 Problem 1

3.1 Problem 1 (a)

The matrix form of this Hamiltonian can be obtained by comparing explicitly in calculation

$$\begin{pmatrix} b_0^\dagger & b_1^\dagger & \dots & b_{L-1}^\dagger & b_\odot^\dagger \end{pmatrix} \begin{pmatrix} M_{0,0} & M_{0,1} & \dots & M_{0,L-1} & M_{0,L} \\ M_{1,0} & M_{1,1} & \dots & M_{1,L-1} & M_{1,L} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ M_{L-1,0} & M_{L-1,1} & \dots & M_{L-1,L-1} & M_{L-1,L} \\ M_{L,0} & M_{L,1} & \dots & M_{L,L-1} & M_{L,L} \end{pmatrix} \begin{pmatrix} b_0^\dagger \\ b_1^\dagger \\ \vdots \\ b_{L-1}^\dagger \\ b_\odot^\dagger \end{pmatrix} \tag{4}$$

$$= \underbrace{\sum_{i=0}^{L-1} b_i^\dagger M_{ii} b_i}_{\text{diagonal}} + \underbrace{\sum_{i \neq j}^{L-2} b_i^\dagger M_{ij} b_j}_{\text{rest}} + \underbrace{\sum_{i=0}^{L-2} b_i^\dagger M_{iL} b_\odot}_{\text{rightmost col } \setminus \text{diag}} + \underbrace{\sum_{i=0}^{L-2} b_\odot^\dagger M_{Li} b_i}_{\text{bot row } \setminus \text{diag}} \tag{5}$$

Since the operators on different sites commute, the matrix entries can be read off the Hamiltonian as

$$\begin{pmatrix} 0 & -t & \dots & -t & -s \\ -t & 0 & \dots & -t & -s \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ -t & -t & \dots & 0 & -s \\ -s & -s & \dots & -s & 0 \end{pmatrix} \tag{6}$$

This $(L+1) \times (L+1)$ matrix is diagonalised for a sample of values of s/t. As previously mentioned in the implementation section, this is done by setting $t = 1$ and varying s, which can be interpreted as expressing the center hopping strength s in units of on-wheel hopping strength t. We obtain a set of eigenvalues for each hopping strength s. The eigenvalues are sorted by magnitude and plotted. The largest and smallest eigenvalue are always coloured individually, all others have the same colour. Figure 3.1 shows the log-linear plot for $L = 2$ up to 11, corresponding to 12 sites in total with the center.

The first observation is the fact that all plots are similar. While the number of eigenvalues increases, there are only three distinct "trajectories": the first and last eigenvalue split up, while all other eigenvalues are constant on a line. When the last eigenvalue and the constant eigenvalues cross, the colours swap since the eigenvalues are ordered by magnitude.

Theoretically, it could be possible to keep track of the eigenvalues correctly by calculating the derivatives at the crossing to identify each eigenvalue. However, it was not immediately clear to us how to do that and so we decided against doing it due to time constraints.

The most important feature is that the splitting increases in magnitude as the number of sites go up. One minor observation is the fact that the first and last eigenvalues start out symmetrically spaced around 0 for $L = 2$, but the ground state drops with increasing L right from the smallest s/t value.

In short, as L increases, the split off of the ground state increases and the initial ground state energy decreases.

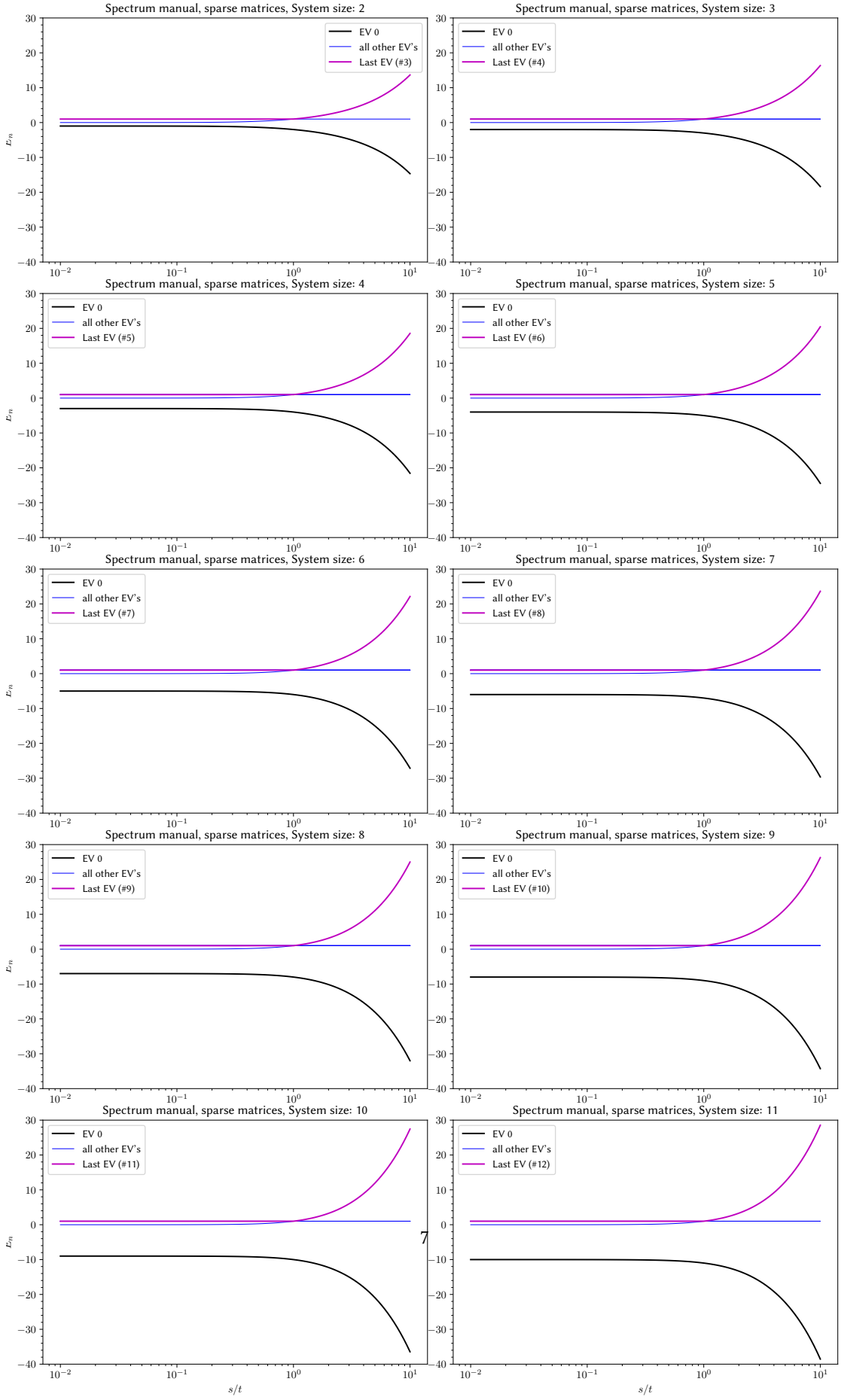

Figure 3.1: Spectrum of singe-particle Hamiltonian for different system sizes

3.2 Problem 1 (b)

It is simple to check that the following identifications from [3] satisfies the commutation relations proposed in the problem statement:

$$\hat{h_i} = \hat{S}_i^+ \qquad \hat{h_i}^\dagger = \hat{S}_i^- \qquad \hat{S}_i^\pm = \hat{\sigma}_i^x \pm \hat{\sigma}_i^y \qquad i \in \{1, ..., L\} \tag{7}$$

We map an "empty site" to $[1, 0] =: |\uparrow\rangle$ and an "occupied site" to $[0, 1] =: |\downarrow\rangle$. The hardcore bosonic ladder operators can then be represented in the Fock basis, e.g. for four sites, $|0011\rangle \hat{=} |\uparrow\rangle\otimes|\uparrow\rangle\otimes|\downarrow\rangle\otimes|\downarrow\rangle$. This fixes the representation of the hardcore bosonic ladder operators in the Fock basis to

$$\hat{h_i} = \hat{S}_i^+ = \underbrace{\mathbb{1}\otimes\mathbb{1}\otimes\cdots}_{\text{i times}}\otimes\hat{S}^+\otimes\underbrace{\cdots\otimes\mathbb{1}\otimes\mathbb{1}}_{\text{L-i times}} \tag{8}$$

With this mapping, the Hamiltonian comes out to be a $2^{L+1}\times 2^{L+1}$ matrix. The correlator matrix takes the same size and can easily be assembled from the hardcore bosonic operators.

3.3 Problem 1 (c)

This matrix is diagonalised for s/t values just like for Problem 1(a) (subsection 3.1). The plot with the eigenvalues for up to $L = 11$ is depicted in Figure 3.3. The first two plots are also shown magnified in Figure 3.2.

The first and last eigenvalues are initially colored differently again, but they are quickly overtaken by the huge bundle of intermediate eigenvalues, which splits up into 3 major branches. Just like before, the color mapping fails at the crossings, resulting in a kink, because it was too difficult to keep track of the numerous eigenvalues (! 4095 at $L = 11$) and their derivatives on the plot. The plot essentially shows the same characteristic as the one in Figure 3.1, with the main difference being the number of Eigenvalues and the spread. The eigenvalues start out constant as well, but additional intermediate eigenvalues start out anywhere between the first and last. Additionally, some of the intermediate eigenvalues also split up and down, essentially forming 3 bundles of eigenvalues. The spread of the bundles increases with the number of sites, and the ground state energy drops for all values of s/t. For $L = 11$, the plot becomes so crowded that even at negligible plotting linewidth, the bundles merge into one.

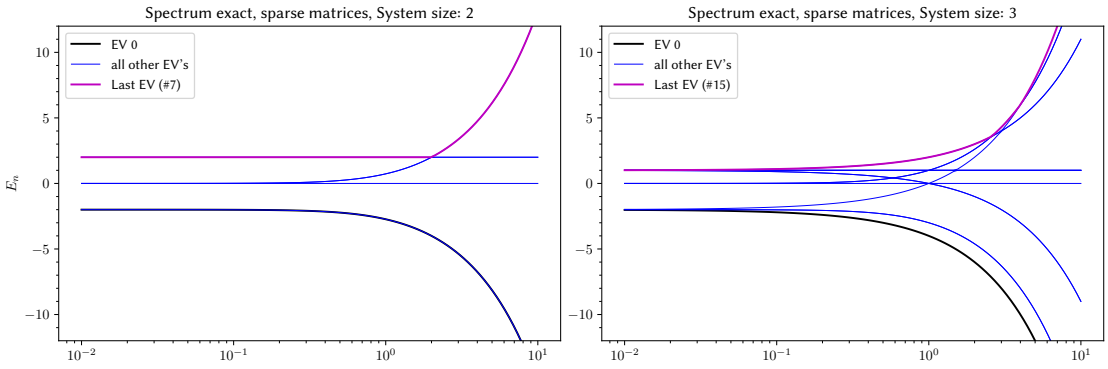

Figure 3.2: Spectrum of Hamiltonian for $L = 2, 3$, which are the first two subplots of Figure 3.3.

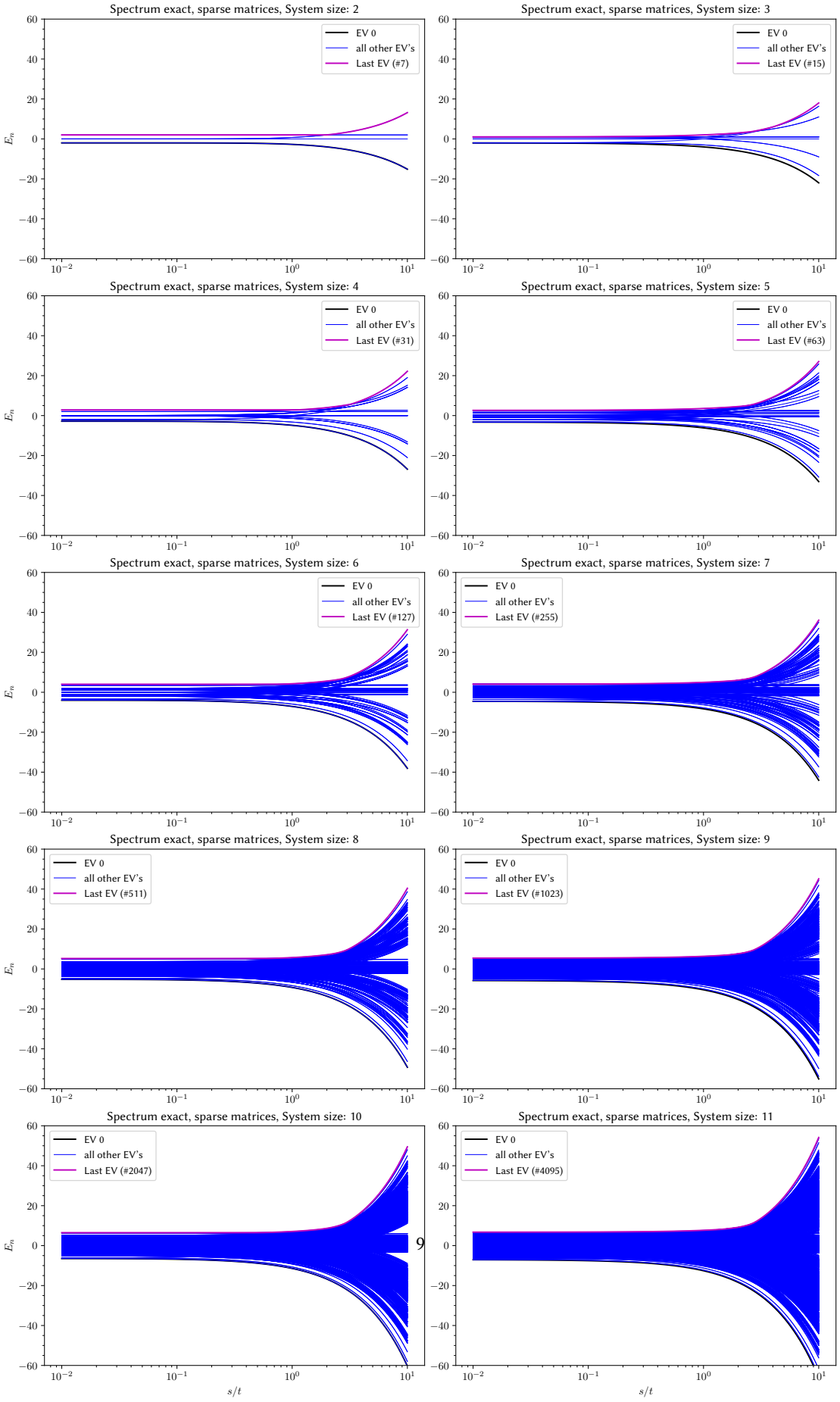

Figure 3.3: Spectrum of Hamiltonian for different system sizes

3.4 Problem 1 (d)

To plot the condensate fraction, the Hamiltonian from Problem 3.2 and 3.3 is diagonalised again. Its ground state is easily picked as the eigenvector corresponding to the (algebraically) smallest (possibly negative) eigenvalue. The expectation value of the correlator matrix $C(i,j)$ from Problem 3.2 is then used to calculate the matrix elements of the reduced density matrix for the given s/t sample.

From the reduced density matrix, the largest eigenvalue n_0 corresponds to the largest number of particles that are correlated in the ground state.

The trace N is equivalent to the number of particles in the ground state, since the diagonal entries of the correlator are just the number operators of a site in the Fock space.

$$N = \sum_j \langle \psi_0 | \, \hat{n}_j \, | \psi_0 \rangle \tag{9}$$

$\rho := \frac{N}{L+1}$ is stored alongside the condensate fraction to store the filling level of the wheel. The condensation fraction is now plotted against $\frac{s}{t}$ in Figure 3.4. The even and odd values of L are separated to not crowd the plot. Additionally, the coloring is picked according to ρ, not L, since there are sometimes multiple trajectories visible for the same L.

We observe that the condensation fractions increase in a sigmoid-like way with increasing s/t. In general, an increase in the condensation fraction for $\frac{s}{t} > 1$ makes sense. In this regime, the hardcore bosons prefer to hop to the center site, which is why they become increasingly correlated, since every site of the wheel is a neighbour to the center. This corresponds nicely with the regime in which the eigenvalues split in Problems 3.1 and 3.3. As the ground state energy gets more negative (the ground state "gets cheaper"), the system's condensation rises. Globally, the even L trajectories are at higher condensation fractions than the odd ones. The $L = 2$ trajectory even stays constant at 1, corresponding to a fully correlated system. Since $\rho = 1/3$ in this case, N must equal 1. Therefore there is only one particle occupying the ground state, which trivially correlates with itself, leading to a trivial condensation fraction.

We observed that all trajectories increase with growing s/t (except for tiny overshoots when settling into constant values and globally constant values). This effect is nicely illustrated on the last page in the supplementary material of [4] in Figure S3. There, the fact that growing s corresponds to higher condensate fractions is nicely illustrated across a wide range of system sizes.

Note that the seemingly uninterrupted lines for $\rho = 1/2$ are just an artefact of the scatter plot marker size in combination with the given resolution in s/t and don't carry any physical significance.

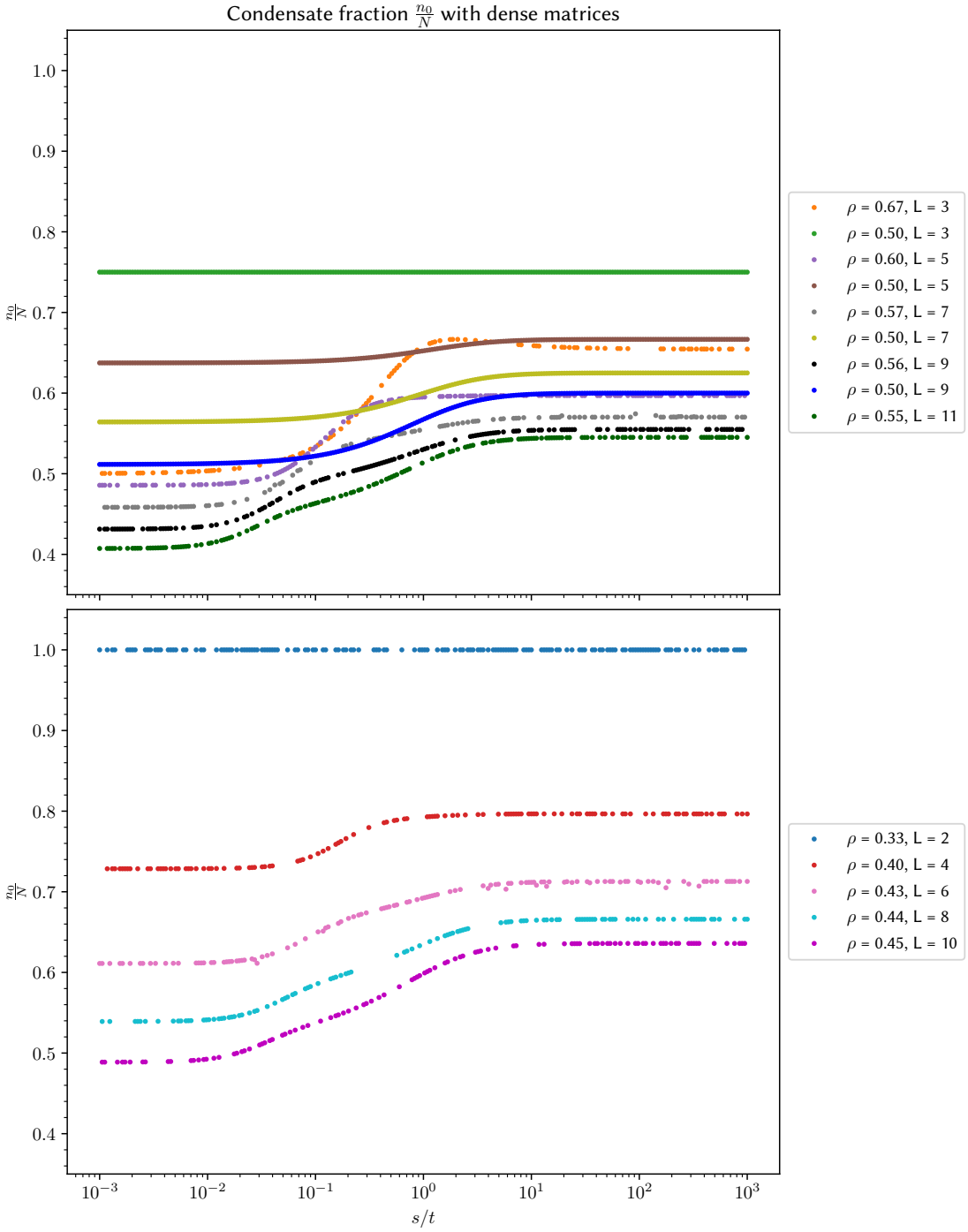

Figure 3.4: Condensate fractions for different number of sites and filling levels

4 Conclusion and Areas for Improvement

In conclusion, we solved the eigenvalue problem for a system size up to $L = 11$ (12 lattice sites in total) in the time that we were given. Unfortunately, on the last day, we noticed an error in the code that meant that we had to rerun all the calculations. Due to time constraints, we were only able to solve for a system size up to $L = 10$ (11 lattice sites in total).

Of course, given the short timeframe, the work done in this project could be improved in many ways, on top of what we have already mentioned within the text.

Firstly, the condensate fraction calculation could have been integrated with the spectrum calculation so that the same Hamiltonian need not be diagonalised twice.

Secondly, The `run.py` script could also have recorded the runtimes of each of the steps, which would have allowed us to visualise the scaling of the problem as a function of lattice size L.

Thirdly, the exact reason why the code does not run on the cluster could be further investigated so that one may have access to much larger computational resources.

Last but not least, we could have also employed the Jordan-Wigner Transformation to reduce the complexity of the problem numerically [4].

References

[1] R. B. Lehoucq, D. C. Sorensen and C. Yang. *ARPACK Users' Guide: Solution of Large Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*. Oct. 1997. (Visited on 21/07/2023).

[2] Haw-Ren Fang and Yousef Saad. "A Filtered Lanczos Procedure for Extreme and Interior Eigenvalue Problems". In: *SIAM Journal on Scientific Computing* 34 (Aug. 2012). DOI: 10.1137/110836535.

[3] Simon Streib and Peter Kopietz. "Hard-core boson approach to the spin $-\frac{1}{2}$ triangular-lattice antiferromagnet Cs_2CuCl_4 at finite temperatures in magnetic fields higher than the saturation field". In: *Phys. Rev. B* 92 (9 Sept. 2015), p. 094442. DOI: 10.1103/PhysRevB.92.094442. URL: https://link.aps.org/doi/10.1103/PhysRevB.92.094442.

[4] R. H. Wilke et al. *Symmetry-Protected Bose-Einstein Condensation of Interacting Hardcore Bosons*. Mar. 2022. DOI: 10.48550/arXiv.2110.15770. arXiv: 2110.15770 [cond-mat, physics:quant-ph]. (Visited on 21/07/2023).